Find The Largest Subtree of Binary Search Tree [GFG](https://practice.geeksforgeeks.org/problems/largest-bst/1)

Given a binary tree. Find the size of its largest subtree that is a Binary Search Tree.  
**Note:**Here Size is equal to the number of nodes in the subtree.

Example 1:

10

/ \

7 17

/ / \

5 15 19

/ /

1. 13

Output: The Largest BST Nodes:

Example 2:

9

/ \

13 6

/ \ / \

16 12 8 1

\ \ \ / \

15 11 7 5 0

Output: The Largest BST Nodes: 1

**Approach 1: Function to find the largest BST subtree within a binary tree using a brute-force approach**

* **Function Purpose:**
  + Find the largest BST subtree within a binary tree using a brute-force approach.
* **Explanation:**
  + Recursively check if the entire tree is a valid BST. If yes, return the size of the entire tree.
  + If the entire tree is not a valid BST, find the largest BST in its left and right subtrees and return the maximum size.
* **Time Complexity:**
  + **In the worst case, we may check all nodes in the tree to validate each subtree.**
  + **Therefore, the time complexity is O(N^2), where N is the number of nodes in the binary tree.**
* **Space Complexity:**
  + **The space complexity depends on the depth of the recursion, which is O(H), where H is the height of the tree.**

**Approach 2: Function to find the largest BST subtree within a binary tree using an optimized approach**

* **Function Purpose:**
  + Find the largest BST subtree within a binary tree using an optimized approach.
* **Explanation:**
  + Perform a bottom-up approach where each node provides information about its subtree.
  + For each node, calculate the size, minimum, and maximum values of the subtree.
  + Check if the subtree is a valid BST, and if so, update the maximum size found so far.
  + Return the maximum size.
* **Time Complexity:**
  + **This approach traverses each node only once, making it an efficient algorithm.**
  + **The time complexity is O(N), where N is the number of nodes in the binary tree.**
* **Space Complexity:**
  + **The space complexity depends on the depth of the recursion, which is O(H), where H is the height of the tree.**

**Conclusion:**

* The optimized approach (Approach 2) is significantly more efficient than the brute-force approach (Approach 1) in terms of time complexity.
* **Approach 2 efficiently calculates the largest BST subtree by avoiding redundant work in checking BST properties for the same subtree multiple times.**
* The optimized approach is preferable for practical use, as it has better performance.